CS6650 ASSIGNMENT 4

Manika Sharma

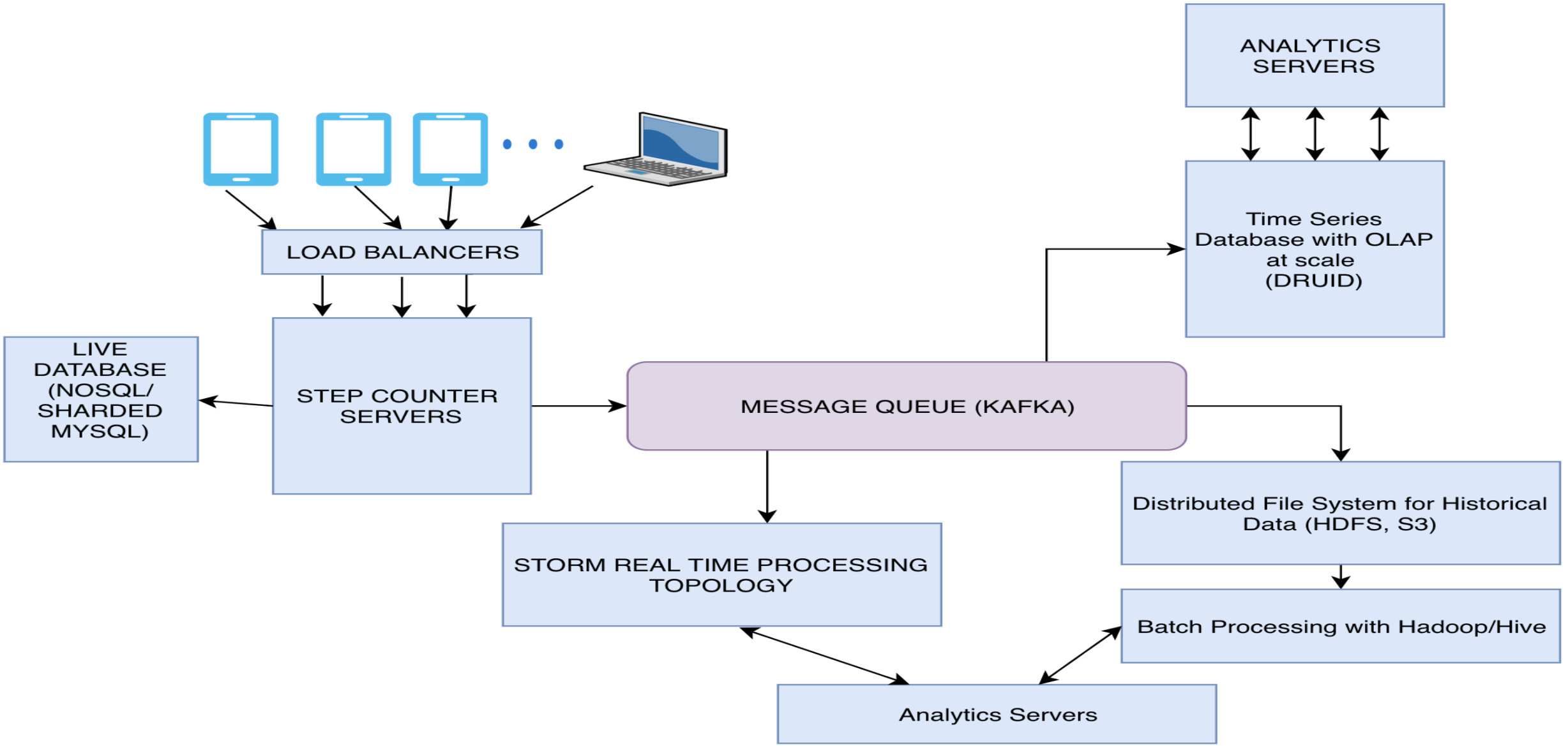
(Kunoichi)

HYPOTHESIS

- Using NoSQL Database to increase transaction (OLTP) throughput.

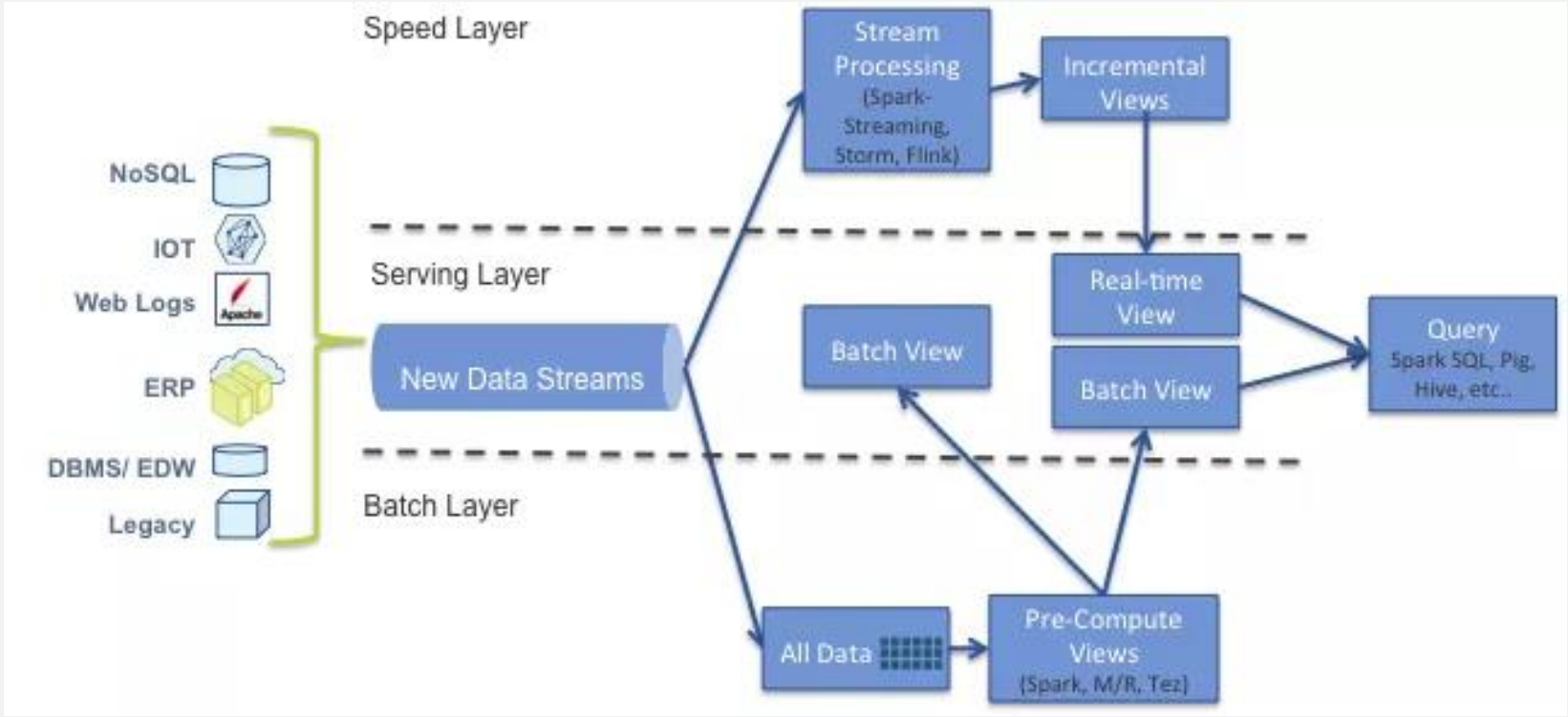
- Using Lambda Architecture to provide analytics (OLAP) at scale using Kafka as message queue and other systems connected to it like hdfs connected to it + hive for batch data processing and storm for real time data processing. Druid is another timeseries database that is built on lambda architecture concepts and provides large scale OLAP over both real-time and historical data.

HIGH LEVEL ARCHITECTURE



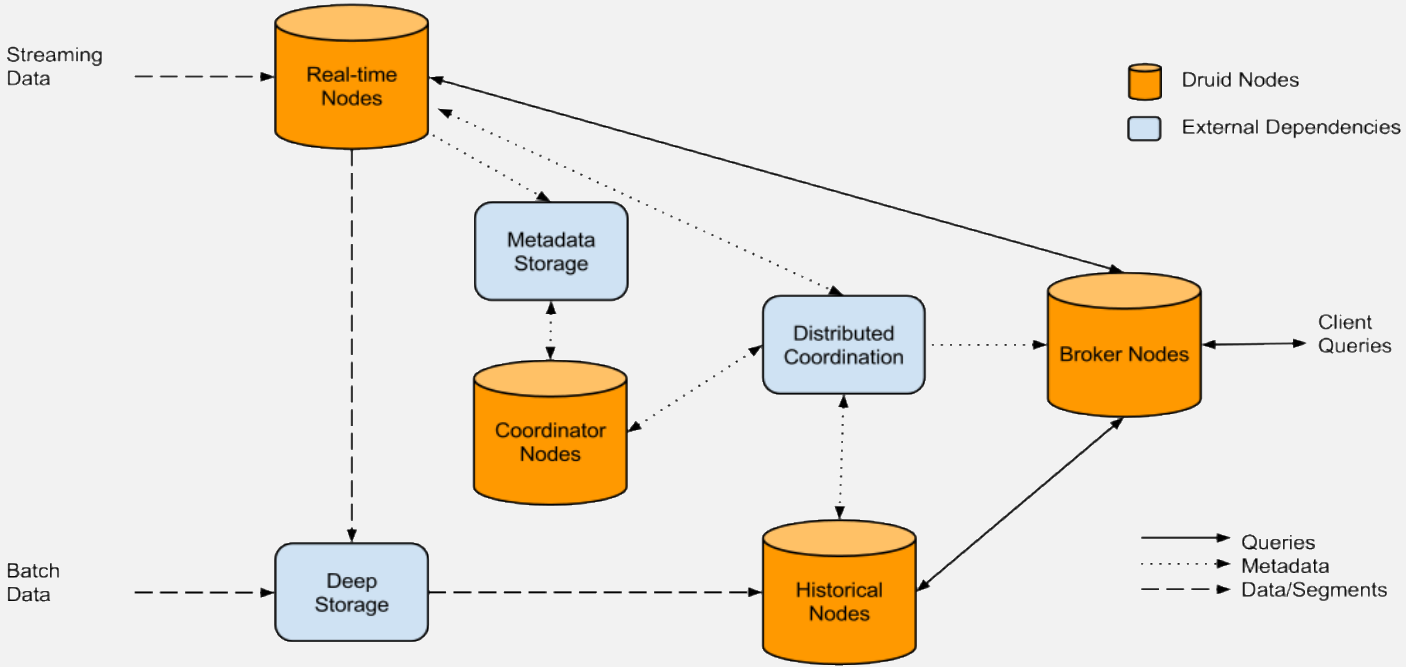
WHAT IS LAMBDA ARCHITECTURE

- Data Processing Architecture designed to handle massive amounts of data by taking advantage of both batch and stream-processing methods



WHY DRUID

- Druid is an open-source data store designed for sub-second queries on real-time and historical data. It is primarily used OLAP queries on event data.
- Druid's core design combines ideas from OLAP/Analytic Databases, timeseries databases and search systems to create a unified system for operational analytics.



HIGH LEVEL OVERVIEW OF WORK INVOLVED

- Clients send calls similar to assignment to Step Counter Servers which are behind load balancer.
- Step Counter Server sends client read/write requests to NoSQL or sharded mysql database to get higher transactional throughput (especially for writes) compared to just one mysql RDS master (with read replicas).
- Step Counter Server also publishes write requests info (with more user dimensions like country, gender etc.) to a kafka queue topic which is partitioned and replicated.
- Druid ingestion servers (realtime nodes) are polling those write events from kafka topic partitions and making segments of data at minute granularity. Those are flushed to hdfs after some time and are queriable by historical nodes by brokers.
- Another option is to attach storm to kafka topic to provide realtime data processing capabilities along with periodically flush of data from kafka to hdfs which is available for batch queries via Hadoop/Hive.
- UI/Graph is exposed via analytics server where one can slice/dice on some user dimensions in a given timeseries table. Eg. Avg steps per given hour of day fvariation across gender/country for past 3-4 months.

PART TWO

MODIFICATIONS

1. Using Managed DynamoDB rather than setting up clusters for Cassandra which is a tedious process.
2. Data now has some dimensional data for getting sensible output from OLAP queries. It has Gender, State along with day, time, userid and step counts.
3. For OLAP, no need for HDFS/ Storm. Druid itself manages historical data in S3 and realtime in middle manager nodes and provides efficient large scale OLAP.

DATA MODEL DESIGN

- User id of DynamoDB is partition Key and Sort Key is day/timestamp in format %05d/%02d
- User info is picked from another metadata service (eg. RDS) which contains details like gender, state etc. This is just a static file in jar as data is very small and static (doesn't change).
- Queries work as follows
  - Put is just normal insert query. Eg. (userId = 1, day=23, time=3, stepcount=400) partition key is userId, sort key is 00023/03 and rest of data is stored directly along with userInfoMap.
  - Get steps for day is simle range scan query Eg. (userId = 1, day = 23) range scan query for userId = 1 and day_ts between 00023/00 and 00023/24
  - Get steps between days also works similar to get steps for day. Eg. (userId = 1, day = 23, numDays = 2) range scan query for userId = 1 and day_ts between 00023/00 and 00024/24
  - Get current day query is little tricky and we paginate backwards from last day for that. Eg. (userId = 1) range scan query in descending order for userId = 1 and day_ts <= (99999/24) and limit results to 24 as we can only have 24 hours in a day, break when first seend day changes.

BENEFITS OF USING DYNAMODB

- Since data is partitioned by userId on multiple shards, we can get infinite horizontal scalability if data is properly modelled and read/write throughput is provisioned correctly.
- Our scans are also efficient as queries usually come for userId and given sequential time range which is also sequentially managed (on disk) by Dynamodb within a shard.
- Increasing load generated from client (increasing threads, instances) is not having any negative effect on latency as long as it stays within provisioned throughput.

DYNAMODB RESULTS

(PROVISIONED READ/WRITE THROUGPUT = 10 UNITS)

16 Threads

```
Client starting.... Time: 1544449268467
WARMUP: All threads(1) running....
WARMUP complete: Time 1.965 seconds
LOADING: All threads(8) running....
LOADING complete: Time 4.192 seconds
PEAK: All threads(16) running....
PEAK complete: Time 12.227 seconds
COOLDOWN: All threads(4) running....
COOLDOWN complete: Time 2.671 seconds
========================================
Total number of requests sent: 119500
Total number of Successful responses: 0
Test Wall Time: 21.056 seconds
Overall throughput across all phases: 5675.341945288754 rps.
P95 Latency = 0 ms.
P99 Latency = 0 ms.
```

32 Threads

```
Client starting.... Time: 1544449350746
WARMUP: All threads(3) running....
WARMUP complete: Time 1.899 seconds
LOADING: All threads(16) running....
LOADING complete: Time 6.544 seconds
PEAK: All threads(32) running....
PEAK complete: Time 21.989 seconds
COOLDOWN: All threads(8) running....
COOLDOWN complete: Time 2.704 seconds
========================================
Total number of requests sent: 240500
Total number of Successful responses: 0
Test Wall Time: 33.138 seconds
Overall throughput across all phases: 7257.529120646992 rps.
P95 Latency = 0 ms.
P99 Latency = 0 ms.
```

DYNAMODB RESULTS

64 Threads

```
Client starting.... Time: 1544449445616
WARMUP: All threads(6) running....
WARMUP complete: Time 2.111 seconds
LOADING: All threads(32) running....
LOADING complete: Time 11.331 seconds
PEAK: All threads(64) running....
PEAK complete: Time 45.608 seconds
COOLDOWN: All threads(16) running....
COOLDOWN complete: Time 4.383 seconds
========================================
Total number of requests sent: 481000
Total number of Successful responses: 0
Test Wall Time: 63.436 seconds
Overall throughput across all phases: 7582.445299199193 rps.
P95 Latency = 0 ms.
P99 Latency = 0 ms.
```

128 Threads

```
Client starting.... Time: 1544449571672
WARMUP: All threads(12) running....
WARMUP complete: Time 3.793 seconds
LOADING: All threads(64) running....
LOADING complete: Time 23.975 seconds
PEAK: All threads(128) running....
PEAK complete: Time 90.133 seconds
COOLDOWN: All threads(32) running....
COOLDOWN complete: Time 7.265 seconds
========================================
Total number of requests sent: 962000
Total number of Successful responses: 0
Test Wall Time: 125.169 seconds
Overall throughput across all phases: 7685.609056555537 rps.
P95 Latency = 0 ms.
P99 Latency = 0 ms.
```

DYNAMODB

256 Threads

```
Client starting.... Time: 1544449761425
WARMUP: All threads(25) running....
WARMUP complete: Time 6.724 seconds
LOADING: All threads(128) running....
LOADING complete: Time 46.477 seconds
PEAK: All threads(256) running....
PEAK complete: Time 184.046 seconds
COOLDOWN: All threads(64) running....
COOLDOWN complete: Time 12.103 seconds
========================================
Total number of requests sent: 1925500
Total number of Successful responses: 0
Test Wall Time: 249.353 seconds
Overall throughput across all phases: 7721.9844958753
P95 Latency = 0 ms.
P99 Latency = 0 ms.
```

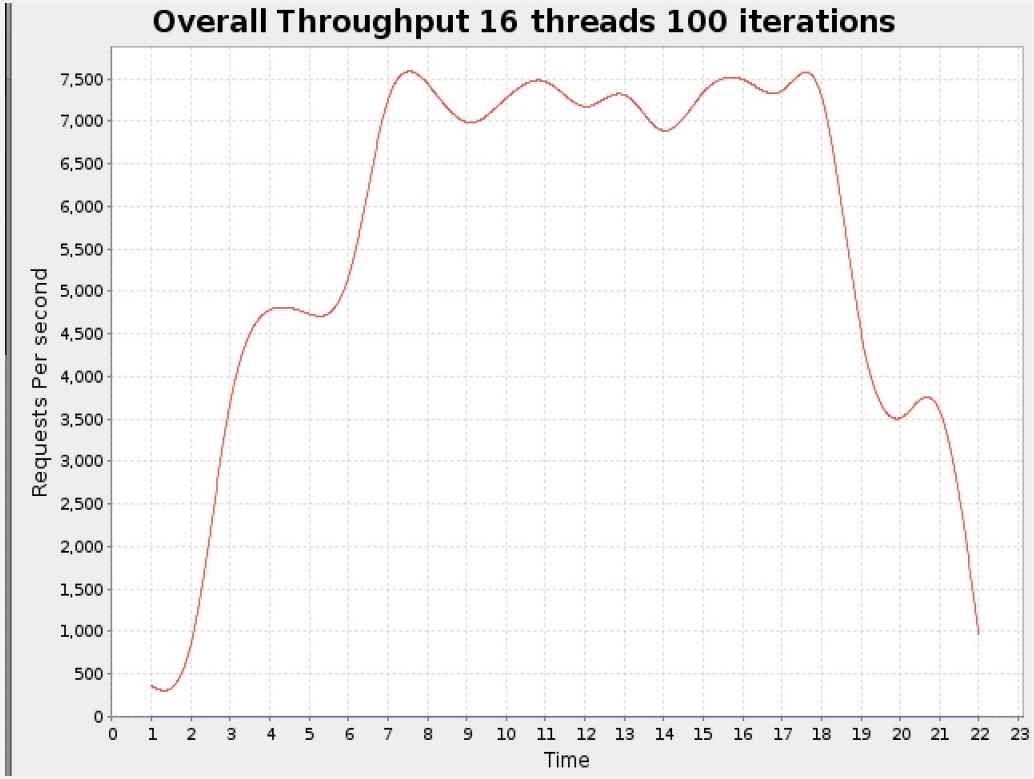
512 Threads

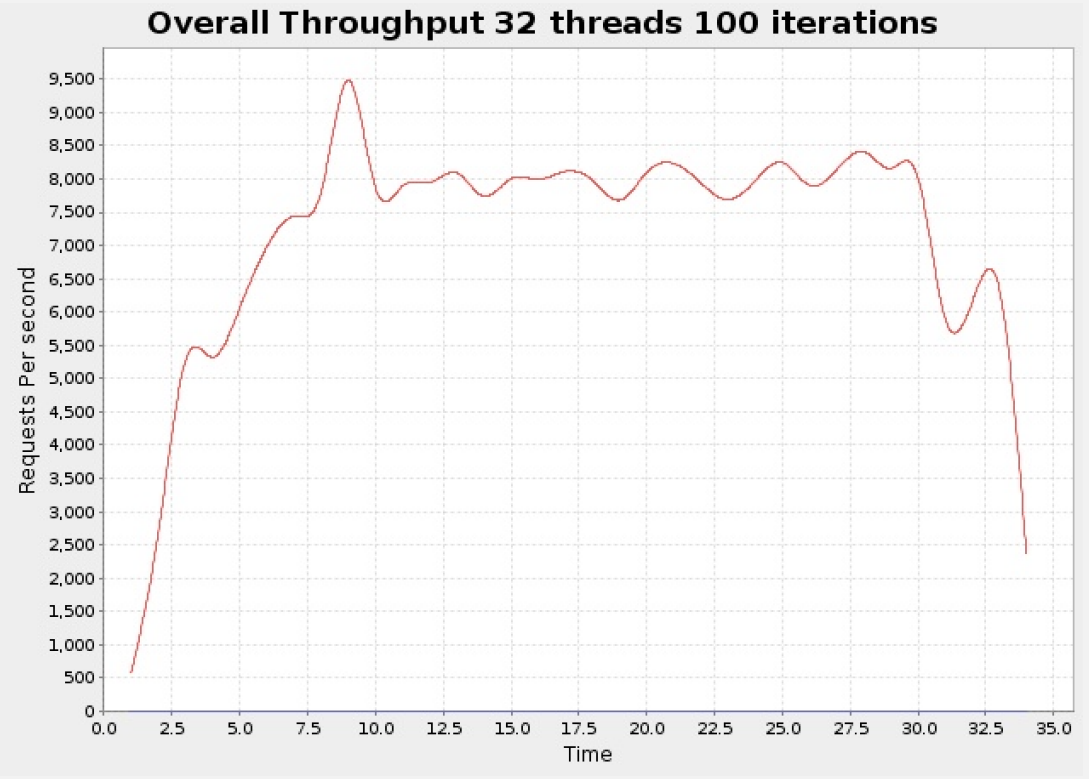
```
Client starting.... Time: 1544450079194
WARMUP: All threads(51) running....
WARMUP complete: Time 12.499 seconds
LOADING: All threads(256) running....
LOADING complete: Time 93.245 seconds
PEAK: All threads(512) running....
PEAK complete: Time 374.394 seconds
COOLDOWN: All threads(128) running....
COOLDOWN complete: Time 24.712 seconds
========================================
Total number of requests sent: 3852500
Total number of Successful responses: 0
Test Wall Time: 504.853 seconds
Overall throughput across all phases: 7630
P95 Latency = 0 ms.
P99 Latency = 0 ms.
```

1024 Threads

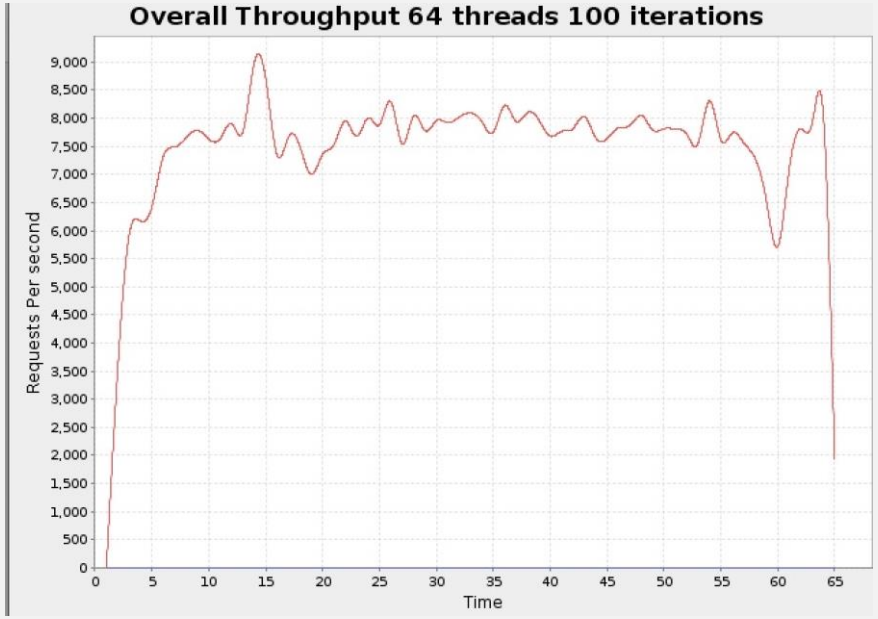
```
Client starting.... Time: 1544450659253
WARMUP: All threads(102) running....
WARMUP complete: Time 24.765 seconds
LOADING: All threads(512) running....
LOADING complete: Time 200.161 seconds
PEAK: All threads(1024) running....
PEAK complete: Time 748.511 seconds
COOLDOWN: All threads(256) running....
COOLDOWN complete: Time 42.586 seconds
========================================
Total number of requests sent: 7705000
Total number of Successful responses: 0
Test Wall Time: 1016.026 seconds
Overall throughput across all phases: 7583.4673
P95 Latency = 0 ms.
P99 Latency = 0 ms.
```

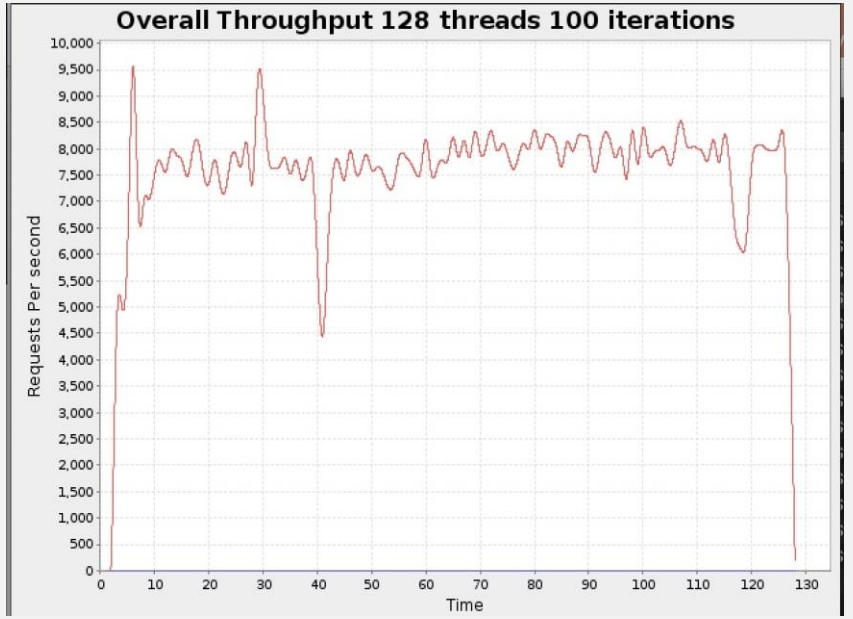
DYNAMODB GRAPHS



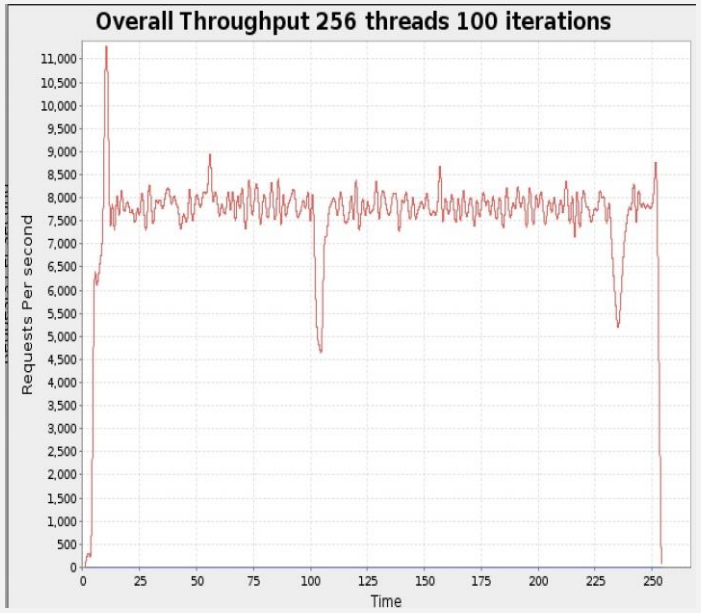


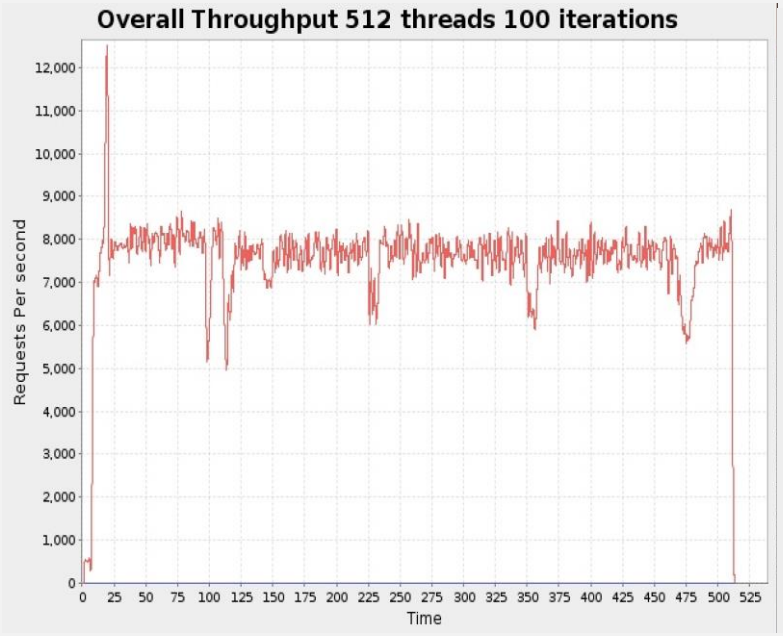
DYNAMODB GRAPHS

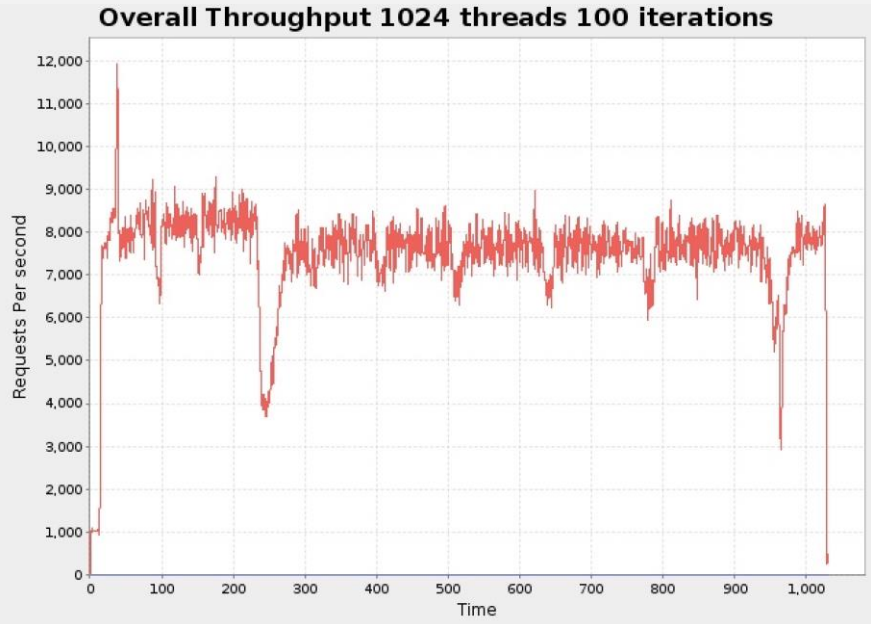




DYNAMODB GRAPHS







I was able to achieve throughput of 1100 RPS while using RDS whereas With DynamoDB, I got almost 8000 RPS thus getting almost 800% Increase in throughput

EVENT PIPELINING

- Server logs all write events(with user info) to Kafka so other systems can asynchronously poll events for batch /stream pipelining at their own speed.
- Configuration of Kafka

1. 5 Zookeeper instances
2. 5 Kafka brokers
3. Topics: stepcounter_test_topic 5 partitions, 3 replicas, write ack from leader only.

Default partition on userId

Step Counter servers asynchronously log write events which is buffered as well so it doesn't add latency to user write requests.

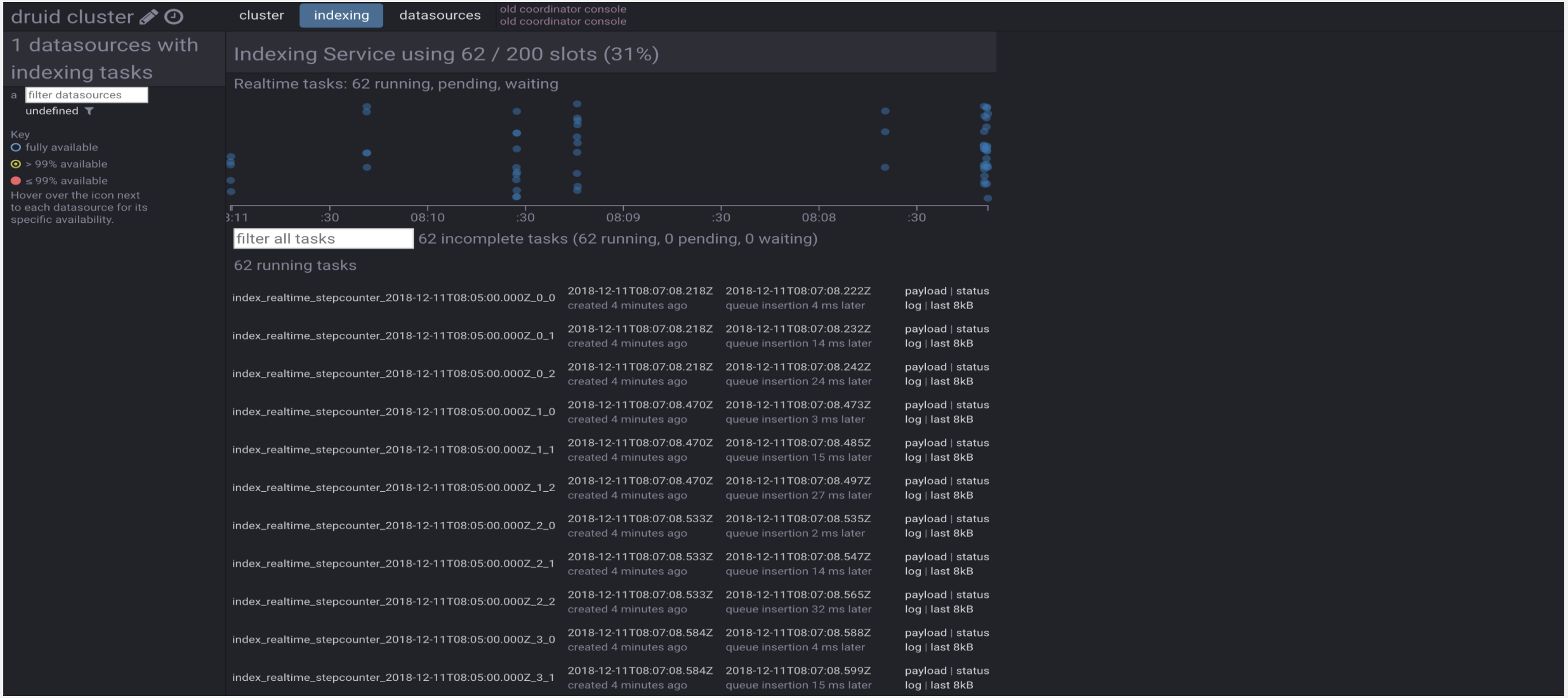
DRUID

- Druid can connect to Kafka using Kafka Indexing service. It polls the events in real time with a spec which tells what are dimensional attributes(state, gender, user) and measure attributes (stepcount, aggregates of stepcount), We use tranquility as Kafka indexing service.
- Realtime data is converted into time segments periodically and then pushed to historical nodes(configured to store on s3) by druid.
- Configuration: 1 coordinator, 1 overload, 3 brokers, 1 historical, 5 middle managers(workers), + 5 tranquility servers (to poll from Kafka) and 5 zookeeper instances (which manages kafka cluster as well)
- Used mysql for medtadata storage about segments and S3 for historical data as well as logs.

Clients/Analytics tool query brokers with time range and olap dimensions and it knows how to contact coordinator, realtime and historical nodes via zookeeper.

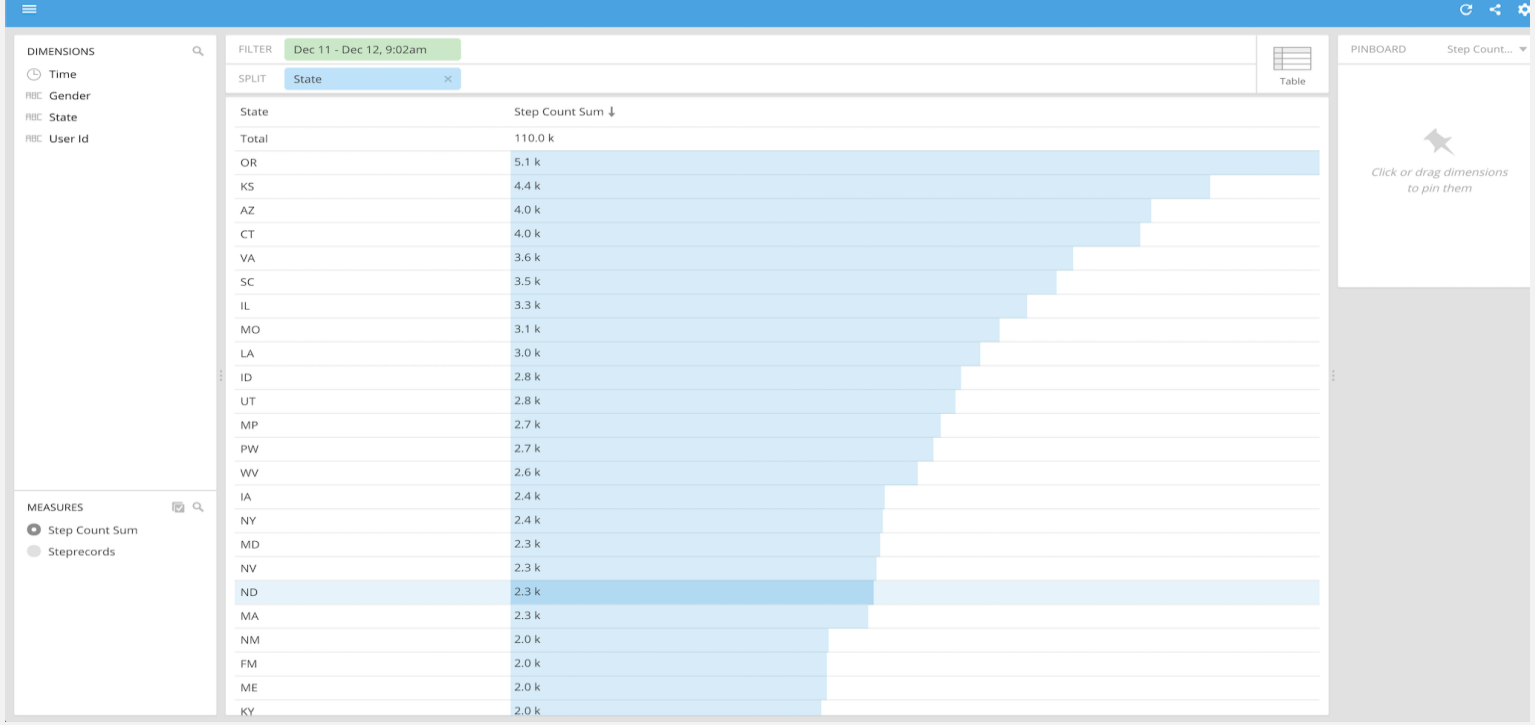
I used Turnilo(previously pivot) which is specifically made for druid integration and provides nice UI to do OLAP queries by click&drag dimensions, selecting time range and measure attributes.

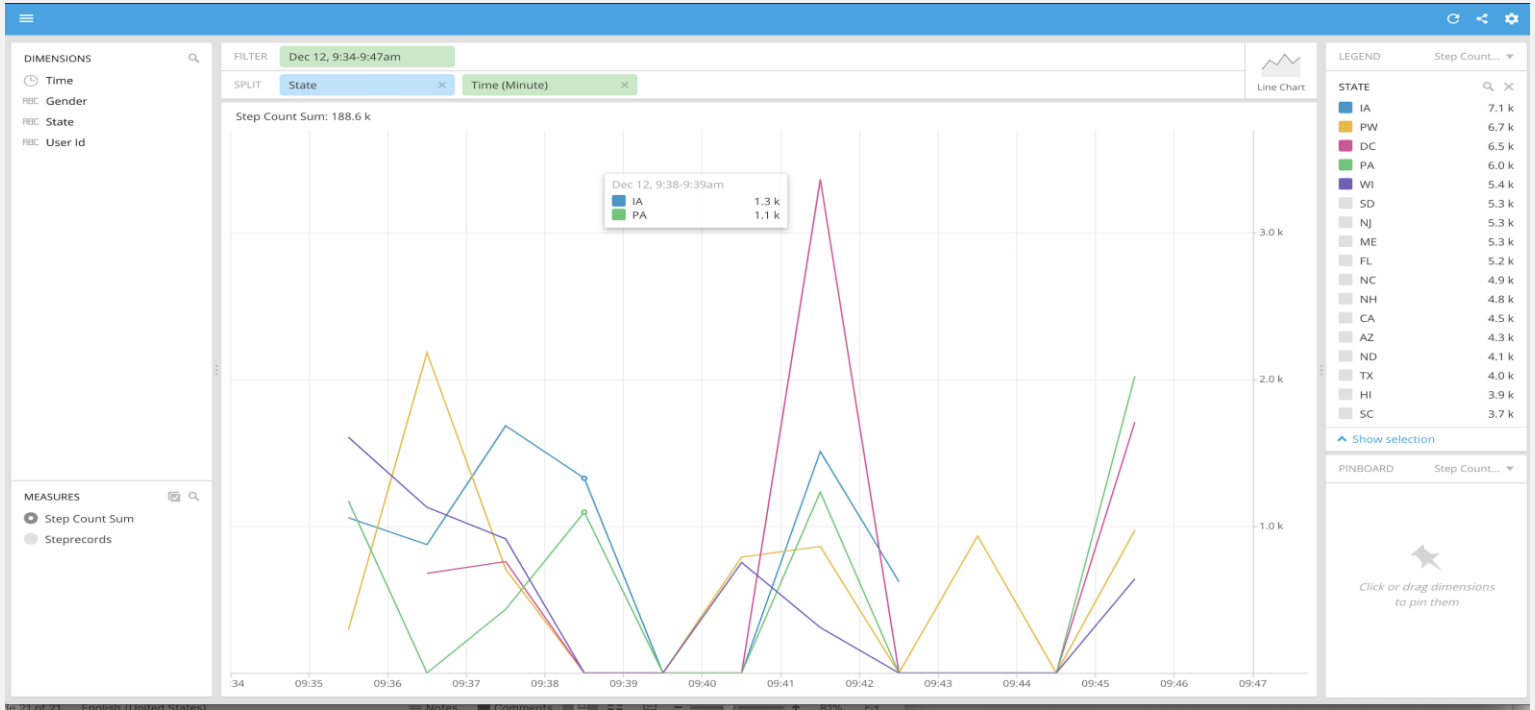
DRUID REAL TIME INDEXER JOBS RUNNING



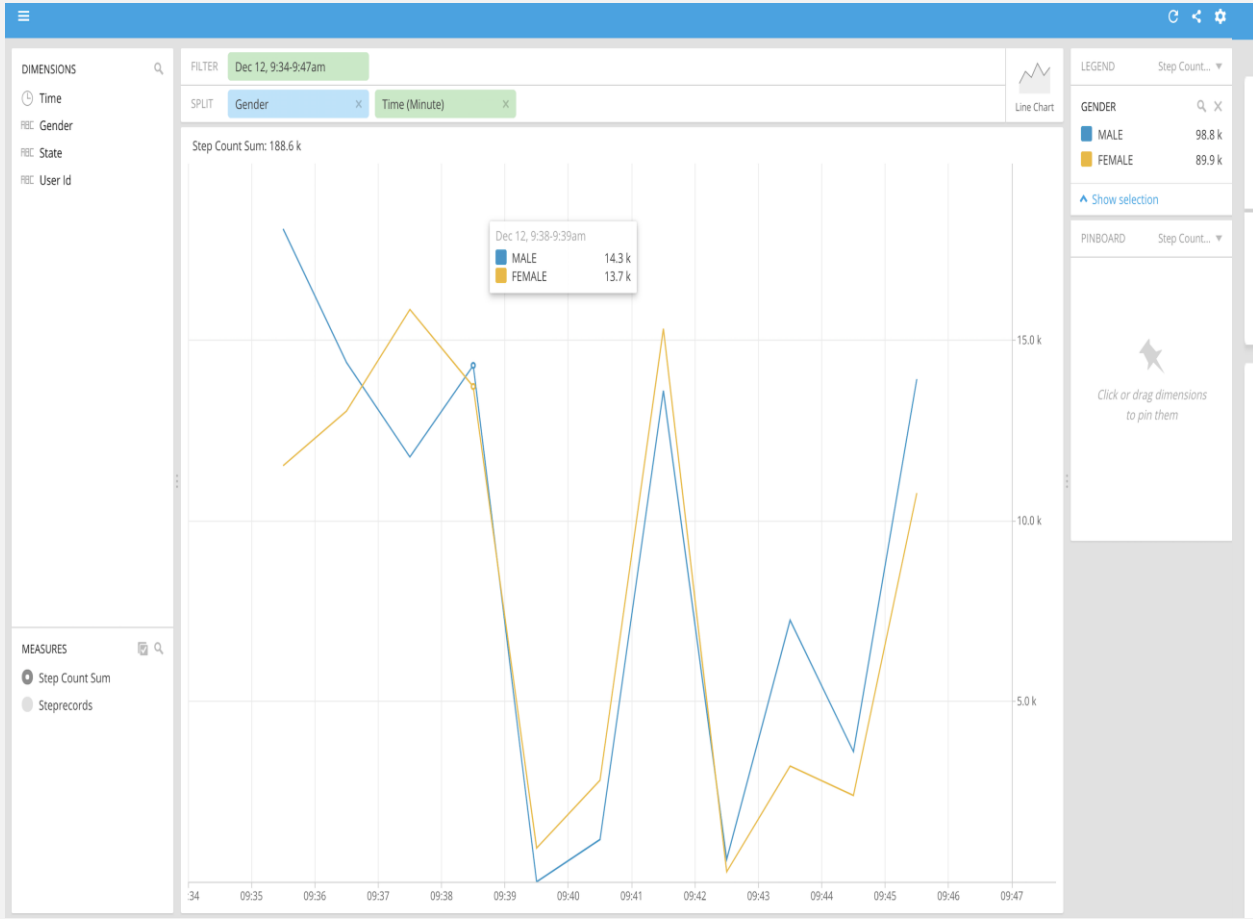
DRUID OLAP IN ACTION

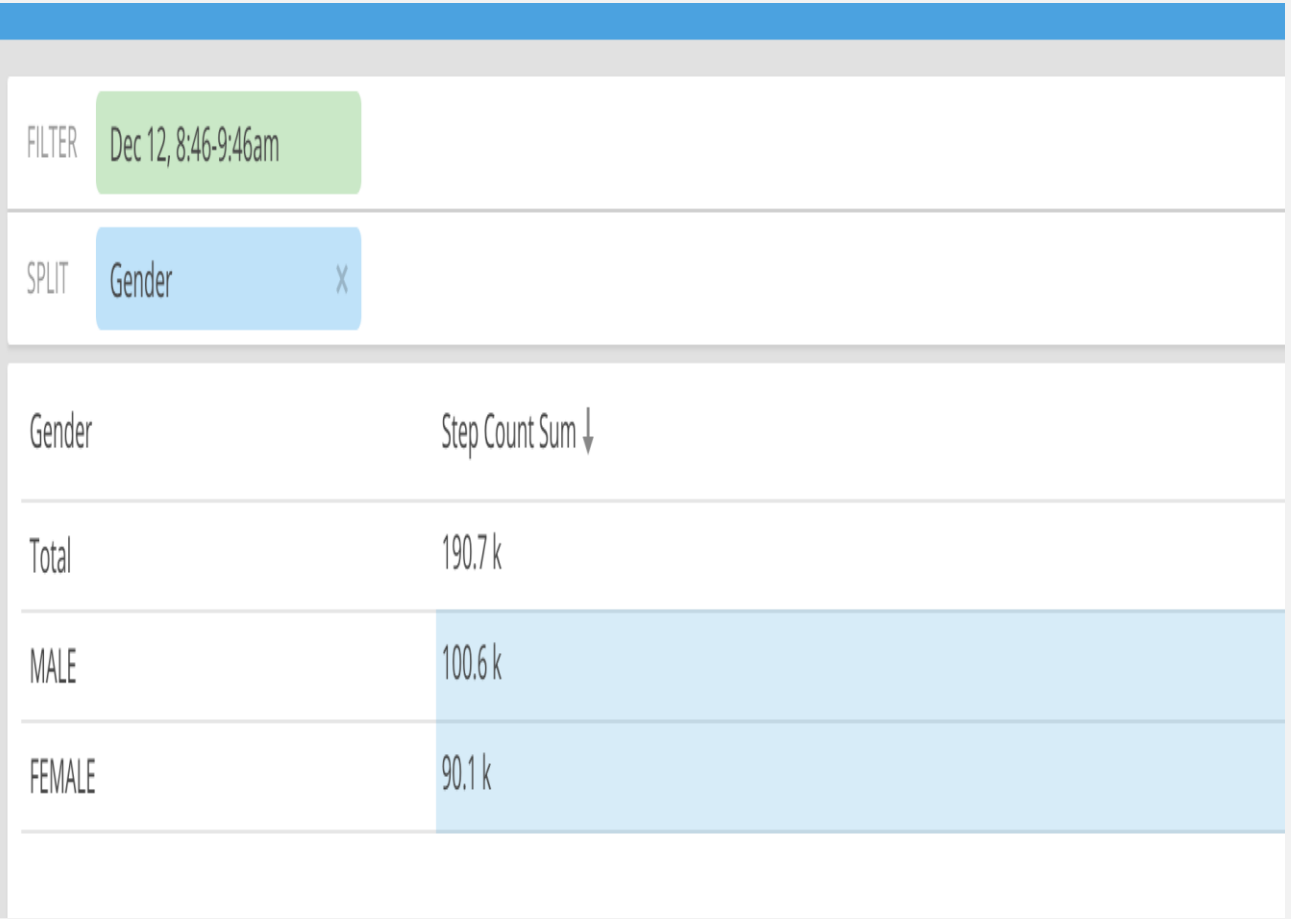
Step Counts According to State





OLAP (STEP COUNTS SPLIT ON GENDER)





CONCLUSION

- In today's world both aspects of data processing are important.
  - Good OLTP system provides for better user experience (low latency, scale, resilience). In our case single RDS instance was not enough to scale write traffic as there was only one master. Dynamodb provided horizontal scaling by partitioning user data across different shards in such a way that each query is limited to one shard (for efficient scan/transaction per user etc.) but many of such queries can run parallelly on multiple shards.
  - Good OLAP system provides better insights (fast, up-to-date, scale, volume) for user data so decisions can be made to target different type of users in different way or to do timeseries analysis of growth, sales etc. Druid provided efficient, fast and up-to-date OLAP at scale by separating but involving both batch and realtime data processing (Lambda Architecture!!).
- Kafka as message queue enables system designers to have one source of truth for all events so different components can be attached to it for different types of processing (batch, realtime, data pipelining etc.) without affecting much on user side frontend servers. Eg. I tried connecting kafka to amazon kinesis, redshift, big table, s3 etc for experimenting with cloud warehouse products using available connectors and it worked seamlessly.
- Using managed services on public cloud systems is much easier than manually setting up clusters and manually provisioning/monitoring them for availability and auto scaling. Initial version of prototypes should be built on top of cloud systems to do faster iterations and verification of system design/performance. If there are some performance concerns or must required features missing, then only custom components should be moved in as replacement to make dev life easier ☺.
- So Yes, My Hypothesis was supported as I was able to get Better Throughput by changing to NoSQL Database and Druid really helped solve OLAP Queries by using up Lambda Architecture.